Regressão

Inteligência na Web e Big Data

Fabricio Olivetti de França e Thiago Ferreira Covões
folivetti@ufabc.edu.br, thiago.covoes@ufabc.edu.br

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

CMCC
Centro de Matemática, Computação e Cognição
Center for Mathematics, Computation and Cognition

Regressão Linear

Aprendendo com exemplos

- Uma das formas de aprendizado de máquina é definido como um conjunto de dados na forma $\{(\mathbf{x}, y)\}$, com $\mathbf{x} \in \mathbb{R}^d$ um vetor de atributos e $y \in \mathbb{R}$ uma variável alvo, queremos descobrir um mapa entre um certo $\mathbf{x}$ para seu valor y correspondente:

$$f : \mathbf{x} \to y.$$

Aprendendo com exemplos

- Vamos tomar como exemplo um corretor que quer aprender a avaliar o valor de um imóvel.
- Cada imóvel pode ser representado por diversas características como:
  - Metragem quadrada
  - Vagas na garagem
  - Andar
  - Bairro
  - etc.

Aprendendo com exemplos

- Para aprender a avaliar novos imóveis ele investiga pelo classificado do jornal vários exemplos:

$\mathbf{m^2}$	vagas	andar	valor
80	1	2	100.000,00
120	2	5	400.000,00
100	1	10	350.000,00
…	…	…	…

Aprendendo com exemplos

- O objetivo é aprender:

$$f(m^2, \text{vagas}, \text{andar}) = \text{valor},$$

para quaisquer valores de m^2, vagas e andar.

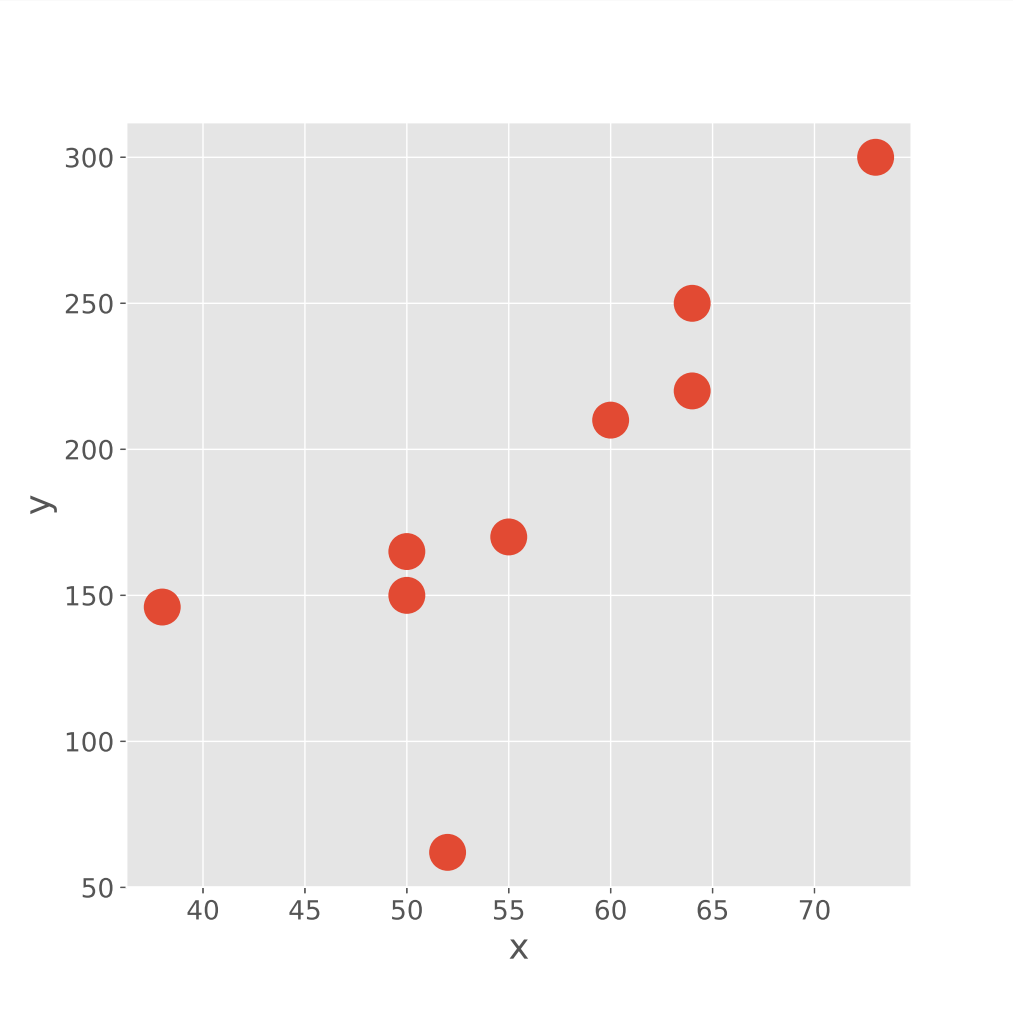
Aprendendo com exemplos

- Reduzindo nossa tabela para duas variáveis, temos:

$\mathbf{m^2}$	**mil** $
52	62
38	146
50	150
50	165
55	170
60	210
64	220
64	250
73	300

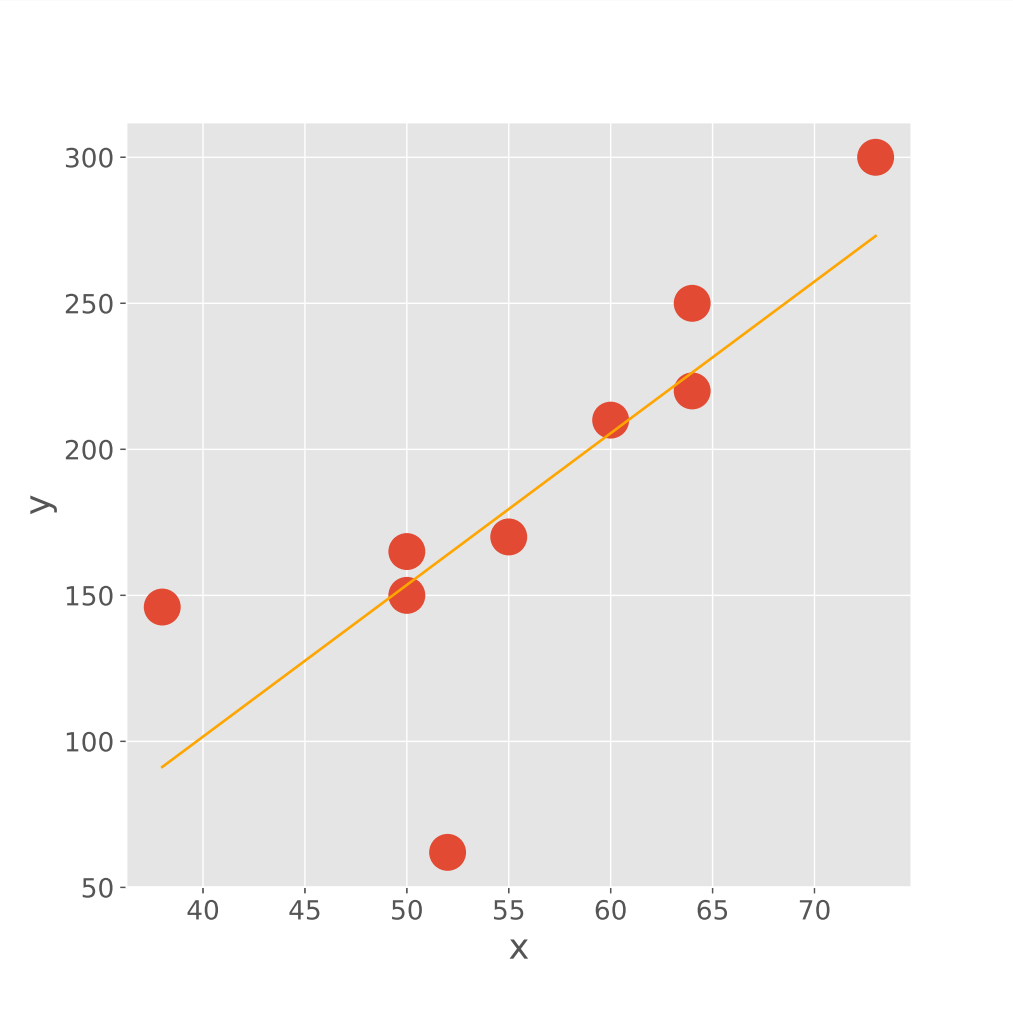
Aprendendo com exemplos

- Com duas variáveis é simples verificar a relação visualmente:



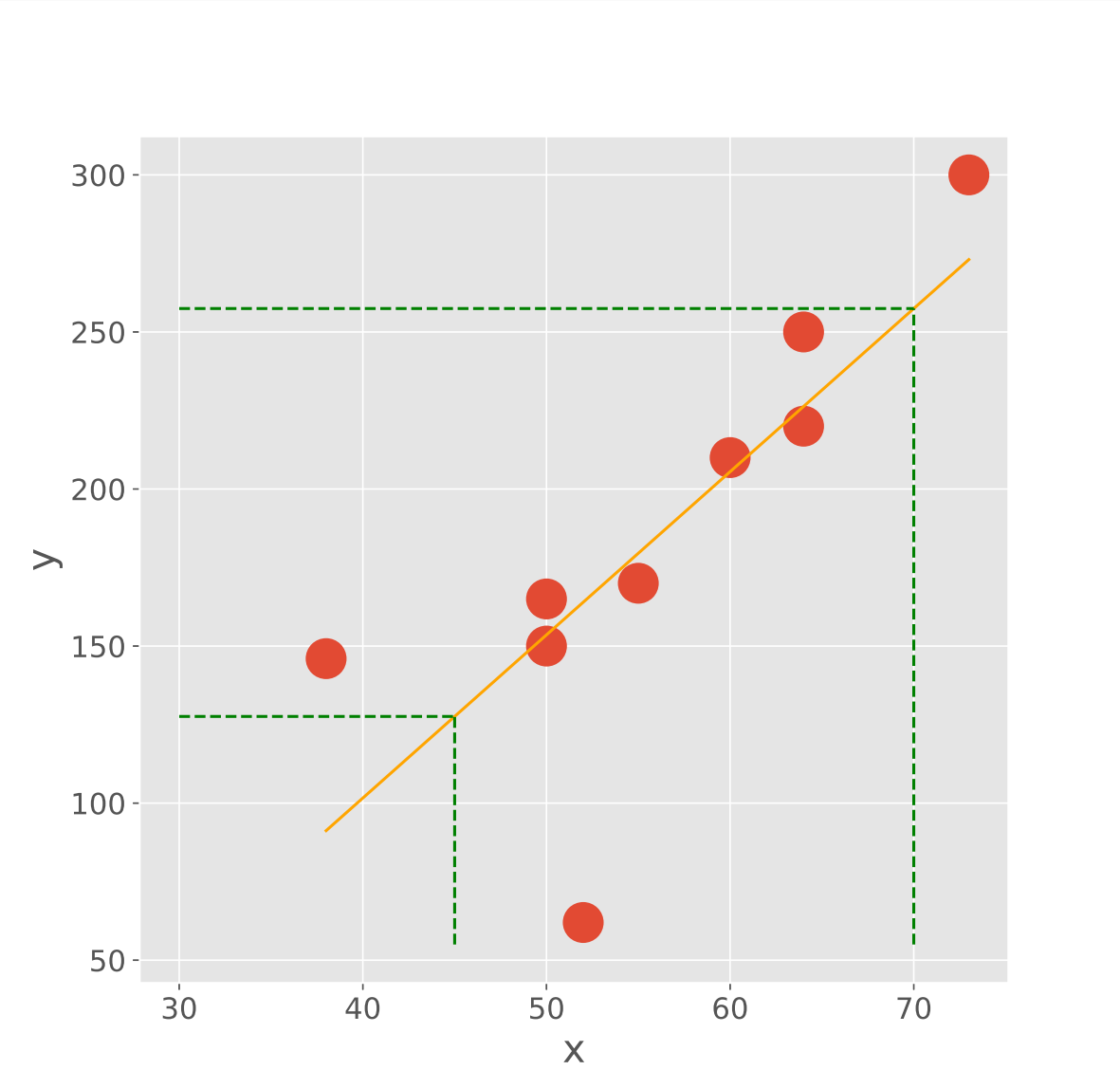
Aprendendo com exemplos

- Com duas variáveis é simples verificar a relação visualmente:



Aprendendo com exemplos

- Com essa reta podemos determinar novos valores:



Regressão Linear

- Esse método é conhecido como regressão linear.
- Ele pode ser usado quando nossas variáveis apresentam uma correlação linear entre elas!
  - mas tem alguns truques para permitir correlações não-lineares também

Regressão Linear

- Dado um conjunto com n exemplos $\{(\mathbf{x}_i, y_i)\}$, com $\mathbf{x}_i \in \mathbb{R}^d, y \in \mathbb{R}$. Queremos encontrar $f : \mathbb{R}^d \to \mathbb{R}$, tal que:

$$f(\mathbf{x}_i) \approx \mathbf{w} \cdot \mathbf{x}_i + b = y_i$$

Regressão Linear

- Renomenado nossas variáveis do exemplo, temos:

$$
\begin{aligned}
x_{i,1} &= m^2 \\
x_{i,2} &= \text{ vagas} \\
x_{i,3} &= \text{ andar} \\
y_i &= \text{ vagas} \\
x_i &= (x_{i,1}, x_{i,2}, x_{i,3})
\end{aligned}
$$

Regressão Linear

$$f(\mathbf{x}_i) = y_i$$

$$\mathbf{w} \cdot \mathbf{x}_i + b = y_i$$

$$w_1 \cdot x_{i,1} + w_2 \cdot x_{i,2} + w_3 \cdot x_{i,3} + b = y_i$$

Regressão Linear

- Definindo $w_0 = b$ e $x_{i,0} = 1$ para todo i, podemos escrever:

$$w_0 \cdot x_{i,0} + w_1 \cdot x_{i,1} + w_2 \cdot x_{i,2} + w_3 \cdot x_{i,3} = y_i$$
$$\mathbf{w} \cdot \mathbf{x}_i = y_i,$$

- ou seja, y é o produto interno entre $\mathbf{w}$ e $\mathbf{x}_i$.

Regressão Linear

- Dados os n exemplos, podemos escrever na forma matricial, $X \in \mathbb{R}^{n \times (d+1)}, \mathbf{y} \in \mathbb{R}^{n \times 1}$:

$$X \cdot w = y,$$

- com $\mathbf{w} \in \mathbb{R}^{(d+1) \times 1}$.
- Esse modelo de regressão é dito interpretável pois é fácil determinar o quanto uma mudança em determinado x_j afeta o valor de y.

Ordinary Least Squares

Ordinary Least Squares

- O nosso problema consiste em determinar $\mathbf{w}$ de tal forma a minimizar o erro de aproximação:

$$e(\mathbf{w}) = (y - X \cdot \mathbf{w})^T (y - X \cdot \mathbf{w}).$$

Ordinary Least Squares

- Para determinar a solução, calculamos a derivada e igualamos a zero para encontrar o mínimo local:

$$\frac{\partial e(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial (y - X \cdot \mathbf{w})^T (y - X \cdot \mathbf{w})}{\partial \mathbf{w}}.$$

Exercício

- Calcule:

$$\frac{\partial e(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial (y - X \cdot \mathbf{w})^T (y - X \cdot \mathbf{w})}{\partial \mathbf{w}}.$$

Exercício

$$\frac{\partial e(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial (y - X \cdot \mathbf{w})^T (y - X \cdot \mathbf{w})}{\partial \mathbf{w}}$$
$$\frac{\partial e(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial (yy^T - (X\mathbf{w})^T y - y^T X\mathbf{w} + (Xw)^T Xw}{\partial \mathbf{w}}$$
$$\frac{\partial e(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial (yy^T - 2\mathbf{w}^T X^T y + \mathbf{w}^T X^T X\mathbf{w}}{\partial \mathbf{w}}$$
$$\frac{\partial e(\mathbf{w})}{\partial \mathbf{w}} = -2X^T y + 2X^T Xw$$

Exercício

$$
\begin{aligned}
-2X^T y + 2X^T X w &= 0 \\
2X^T X w &= 2X^T y \\
X^T X w &= X^T y \\
w &= (X^T X)^{-1} X^T y
\end{aligned}
$$

Ordinary Least Squares

- Temos então que:

$$w = (X^T X)^{-1} X^T y,$$

- com $(X^T X)^{-1} X^T$ sendo a pseudo-inversa de X.

Ordinary Least Squares

- Para dimensões grandes de X o custo da multiplicação de matrizes pode se tornar proibitivo.
- Uma implementação simples, assumindo a existência de uma função inv que inverte uma matriz é dada por:

```haskell
ols :: [[Double]] -> [Double] -> [Double]
ols x y = multMtx pseudoInv (multMtx (transpose x) y)
   where
     pseudoInv = inv $ multMtx (transpose x) x
```

Ordinary Least Squares

- A multiplicação $X^T X$ pode ser simplificada como a soma dos produtos externos de cada x_i por ele mesmo:

```haskell
ols :: [[Double]] -> [Double] -> [Double]
ols x y = multMtx pseudoInv (multMtx (transpose x) y)
   where
     pseudoInv = inv $ sum $ map autoOuterProd x
     autoOuterProd xi = outerProd xi xi
```

Ordinary Least Squares

- Da mesma forma $X^T y$ pode ser simplificada como a soma dos vetores $\mathbf{x}_i \cdot y_i$:

```haskell
ols :: [[Double]] -> [Double] -> [Double]
ols x y = multMtx pseudoInv $ sumVec (y .*.. x)
   where
     pseudoInv = inv $ sum $ map autoOuterProd x
     autoOuterProd xi = outerProd xi xi
```

Ordinary Least Squares

- Em um contexto de computação distribuída, imaginamos que temos pedaços de $(\mathbf{x}_i, y_i)$ espalhados pelo cluster de máquinas. Dessa formas podemos paralelizar pensando em funções de map e reduce:

```
1  mapper :: ([Double], Double) -> ([[Double]], [Double])
2
3  reducer :: ([[Double]], [Double]) -> ([[Double]],
   ↪  [Double]) -> ([[Double]], [Double])
4
5  ols :: ChunksOf ([Double], Double) -> [Double]
6  ols xy = multMtx pseudoInv xTy
7     where
8       xTx, xTy  = mapReduce mapper reducer xy
9       pseudoInv = inv xTx
```

Ordinary Least Squares

- Como implementar as funções mapper e reducer?

```haskell
mapper :: ([Double], Double) -> ([[Double]], [Double])

reducer :: ([[Double]], [Double]) -> ([[Double]],
 ↪  [Double]) -> ([[Double]], [Double])

ols :: ChunksOf ([Double], Double) -> [Double]
ols xy = multMtx pseudoInv xTy
   where
     xTx, xTy  = mapReduce mapper reducer xy
     pseudoInv = inv xTx
```

Ordinary Least Squares

- Como implementar as funções mapper e reducer?

```haskell
mapper :: ([Double], Double) -> ([[Double]], [Double])
mapper (x, y) = (outerProd x x, y *. x)

reducer :: ([[Double]], [Double]) -> ([[Double]],
 ↪  [Double])
        -> ([[Double]], [Double])
reducer (a, b) (a, b) = (a ..+.. b, a .+. b)
```

Gradiente Descendente

Gradiente Descendente

-Uma outra forma de resolver o problema é utilizando métodos de gradiente para otimização.

- Dada uma função $f(x)$ definida e diferenciável em torno de um ponto x_0, sabemos que ela decresce mais rapidamente na direção oposta do gradiente $\nabla f(x_0)$.

Gradiente Descendente

- Ou seja, fazendo:

$$x^{t+1} = x^t - \alpha \cdot \nabla f(x^t).$$

- Temos que $f(x^{t+1}) \leq f(x^t)$, para um α pequeno.

Gradiente Descendente

- Temos então que:

$$f(x^0) \geq f(x^1) \geq f(x^2) \geq f(x^t).$$

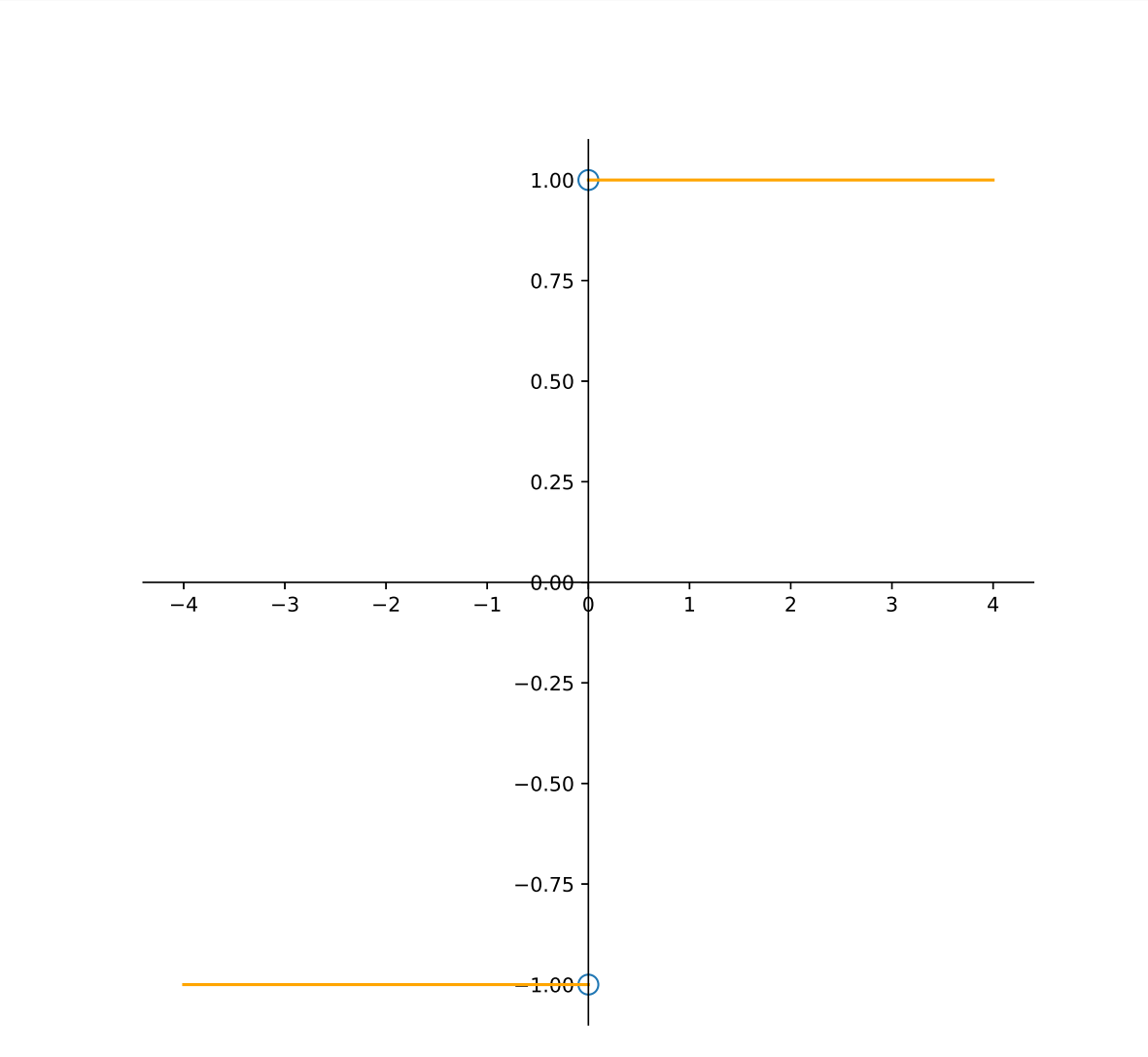
Gradiente Descendente

- Estamos interessados na minimização da aproximação de $f(\mathbf{x})$ com y:

$$e(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \mathbf{x}_i \cdot \mathbf{w}|,$$

Gradiente Descendente

- Porém, a derivada da função valor absoluto é indefinida no ponto 0:

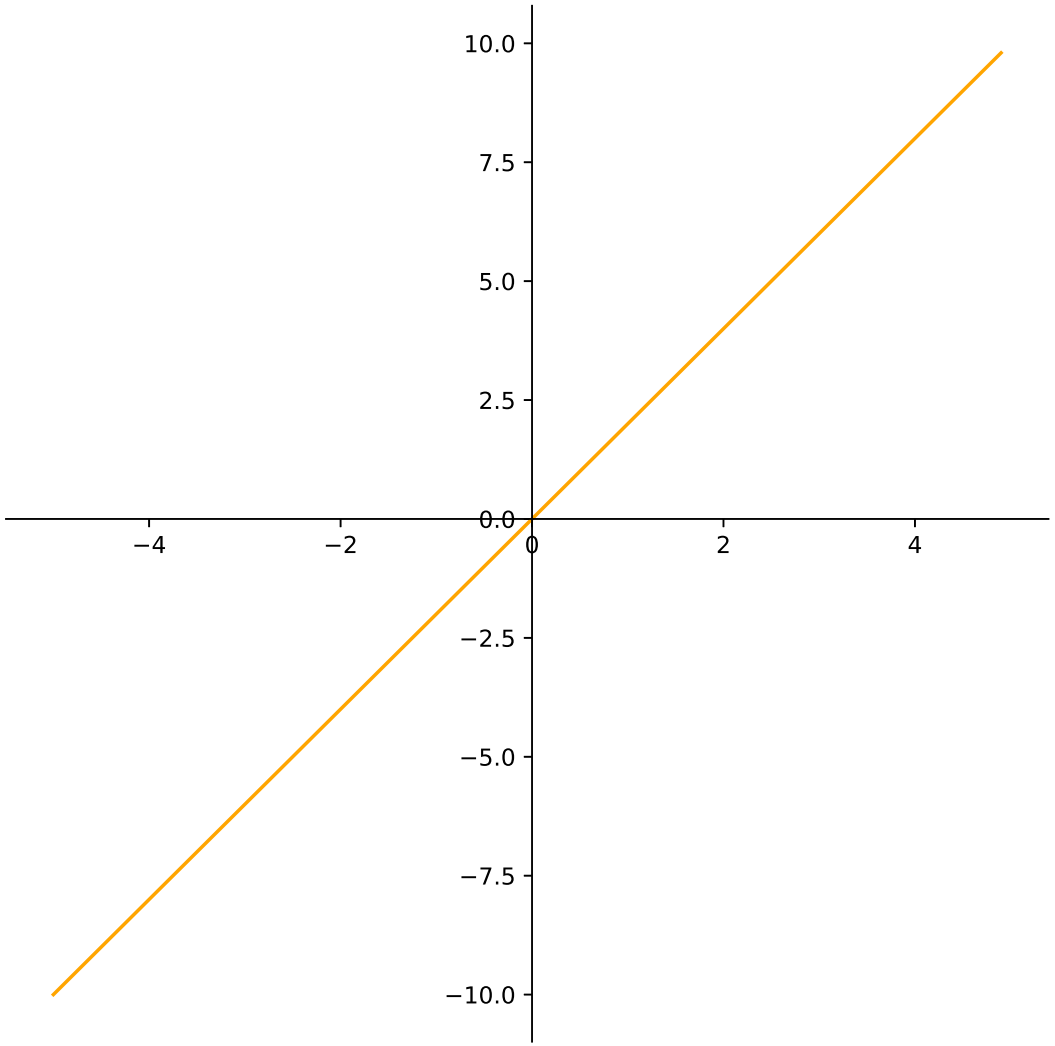


Gradiente Descendente

- Por outro lado, a função quadrática apresenta uma única solução ótima, e é bem definida:

$$e(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{x}_i \cdot \mathbf{w})^2,$$

Gradiente Descendente



Gradiente Descendente

- Temos então que:

$$\nabla e(\mathbf{w}) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i \cdot \mathbf{w}) \cdot x_i,$$

Gradiente Descendente

- Ou com o mesmo ótimo:

$$\nabla e(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i \cdot \mathbf{w}) \cdot x_i,$$

Gradiente Descendente

- O novo valor para $\mathbf{w}$ pode ser calculado como:

$$\mathbf{w^{t+1}} = \mathbf{w^t} + \alpha E((y_i - \mathbf{x}_i \cdot \mathbf{w}) \cdot x_i),$$

Gradiente Descendente

- O algoritmo pode ser descrito como:

```haskell
gradDesc' :: [[Double]] -> [Double] -> [Double]
          -> Double -> Int -> [Double]
gradDesc' x y w alpha it
  | convergiu = w
  | otherwise = gradDesc' x y w' alpha (it-1)
  where
    w'        = w .+. (alpha *. nabla)
    nabla     = mediaVetor
              $ map (\(yi, xi) -> yi *. xi)
              $ zip (y .-. y') x
    y'        = map (dotprod w) x
    convergiu = (erro' < 1e-6) || (it==0)
```

Gradiente Descendente

- Se X é muito grande, temos que paralelizar o cálculo de nabla. Note que o cálculo de y' do qual nabla depende, já está pronto para ser paralelizado.

Gradiente Descendente

- Uma forma de se pensar nessa paralelização é através de um processo mapper e reducer que recebem tuplas de um vetor $\mathbf{x}$ e um escalar y:

```haskell
gradDesc' :: ChunksOf [(Vector Double, Double)]
          -> Vector Double -> Double -> Int
          -> Vector Double
gradDesc' xy w alpha it
  | convergiu = w
  | otherwise = gradDesc' xy w' alpha (it-1)
  where
    w'        = w .+. (alpha *. (nabla ./ n))
    nabla     = mapReduce mapper reducer xy
    convergiu = (it==0)
    n         = sum $ map length' xy
```

Exercício

- Escreva as funções mapper e reducer:

Exercício

- Escreva as funções mapper e reducer:

```haskell
mapper :: ([Double], Double) -> [Double] -> [Double]
mapper (xi, yi) = ( yi - (dotprod w xi) ) *. xi

reducer :: [Double] -> [Double] -> [Double]
reducer = (.+.)
```

Escolha de α

- A escolha do valor de passo de atualização de $\mathbf{w}$ é importante pois um valor muito baixo ocasiona uma demora para atingir o ponto de convergência, por outro lado um valor muito alto faz com que o método do gradiente ultrapasse o ponto de ótimo levando a divergência.

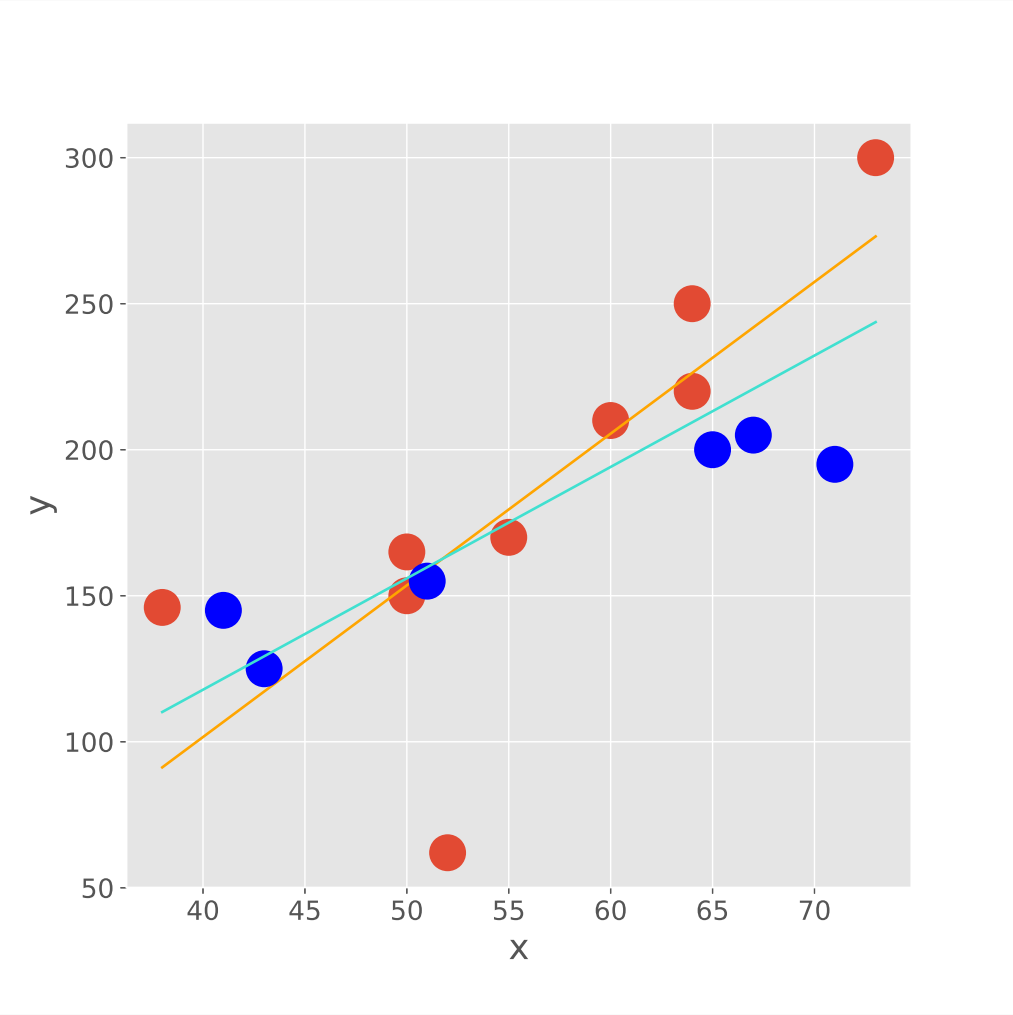
Overfitting

Overfit

- Em muitos casos, a amostra de dados coletada não é representativa. No nosso exemplo, isso pode acontecer ao coletar dados de imóveis de apenas uma região específica.

Overfit

- Isso faz com que a reta de regressão não necessariamente represente a realidade:



Overfit

- Isso pode ocorrer por causa de:
  - Coleta de dados sem o devido cuidado estatístico.
  - Apenas algumas da d variáveis são realmente importantes.
  - Algumas das variáveis são correlacionadas (ex.: área útil e área total).

Regularização

- Para resolver tal problema, utilizamos a regularização na função-objetivo. Para isso alteramos o cálculo do erro para:

$$e(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i \cdot \mathbf{w})^2 + \lambda \sum_{i=0}^{d} |w_i|^p,$$

Regularização

- Com isso a equação de atualização de $\mathbf{w}$ passa a ser calculada como:

$$\mathbf{w^{t+1}} = \mathbf{w^t} + \alpha E((y_i - \mathbf{x}_i \cdot \mathbf{w}) \cdot x_i) - \lambda l_p,$$

- com l_p sendo o gradiente da regularização escolhida.

Regularização l_2

- Para $p = 2$ temos a norma quadrática cujo gradiente resultará em:

$$\mathbf{w^{t+1}} = \mathbf{w^t} + \alpha E((y_i - \mathbf{x}_i \cdot \mathbf{w}) \cdot x_i) - \lambda \mathbf{w}.$$

- Essa é conhecida como regularização l_2 ou ridge. Esse tipo de regularização incentiva baixos valores para todos os elementos do vetor $\mathbf{w}$.

Regularização l_1

- Para $p = 1$ temos o valor absoluto, o que resulta em descontinuidade. Portanto a fórmula de atualização fica:

$$\Delta \mathbf{w^t} = \mathbf{w^t} + \alpha E((y_i - \mathbf{x}_i \cdot \mathbf{w}) \cdot x_i)$$

$$\mathbf{w_i^{t+1}} = \begin{cases} \Delta \mathbf{w_i^t} - \lambda & \text{se } \Delta \mathbf{w_i^t} > \lambda \\ \Delta \mathbf{w_i^t} + \lambda & \text{se } \Delta \mathbf{w_i^t} < -\lambda \\ \Delta \mathbf{w_i^t} & \text{c.c.} \end{cases}$$

- Essa é conhecida como regularização l_1 ou lasso. Esse tipo de regularização incentiva que alguns valores de $\mathbf{w}$ sejam zeros.

Regularização ElasticNet

- Podemos combinar ambas as regularizações:

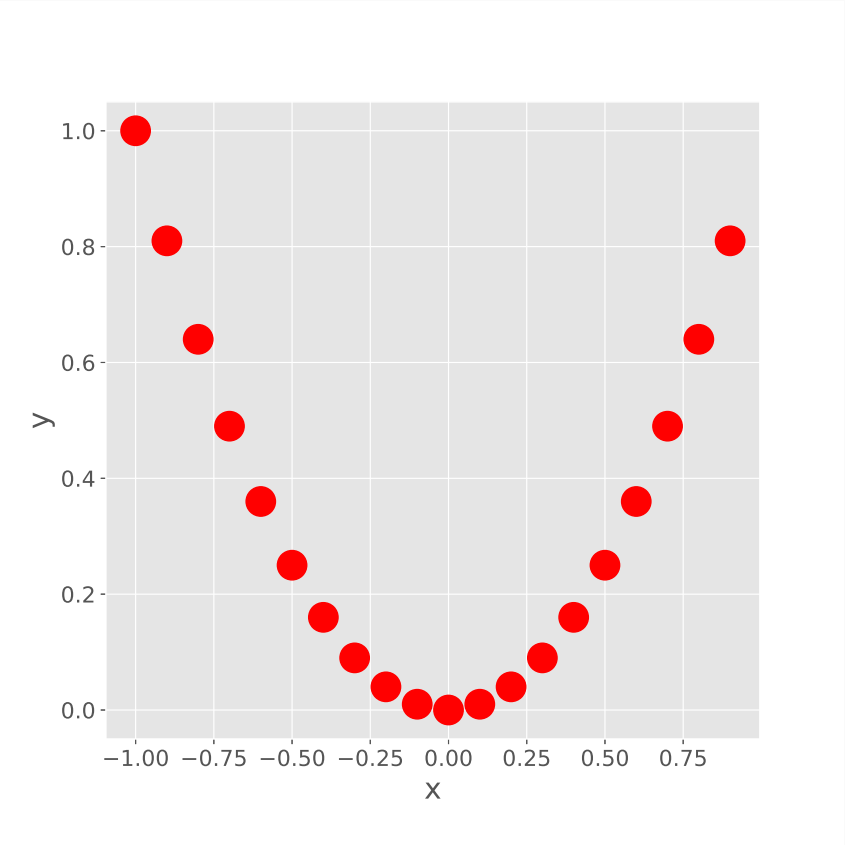
$$\Delta \mathbf{w^t} = \mathbf{w^t} + \alpha E((y_i - \mathbf{x}_i \cdot \mathbf{w}) \cdot x_i)$$

$$\mathbf{w_i^{t+1}} = \begin{cases} \Delta \mathbf{w_i^t} - \lambda_1 - \lambda_2 \mathbf{w} & \text{se } \Delta \mathbf{w_i^t} > \lambda \\ \Delta \mathbf{w_i^t} + \lambda_1 - \lambda_2 \mathbf{w} & \text{se } \Delta \mathbf{w_i^t} < -\lambda \\ \Delta \mathbf{w_i^t} - \lambda_2 \mathbf{w} & \text{c.c.} \end{cases}$$

- Essa é conhecida como regularização ElasticNet.

Variáveis Não-Lineares

Variáveis Não-Lineares



- É fácil perceber que a regressão linear não é um bom modelo para os pontos acima. Qualquer reta utilizada para aproximar os dados irá gerar um erro muito grande em algumas regiões desse gráfico.

Variáveis Não-Lineares

- Uma alternativa para esses casos é criação de novos atributos como funções não lineares.
- Esses novos atributos podem ser:
  - **Transformação:** aplicação de uma função não-linear em uma combinação linear das variáveis.
  - **Interação:** interação polinomial entre duas ou mais variáveis.

Transformação

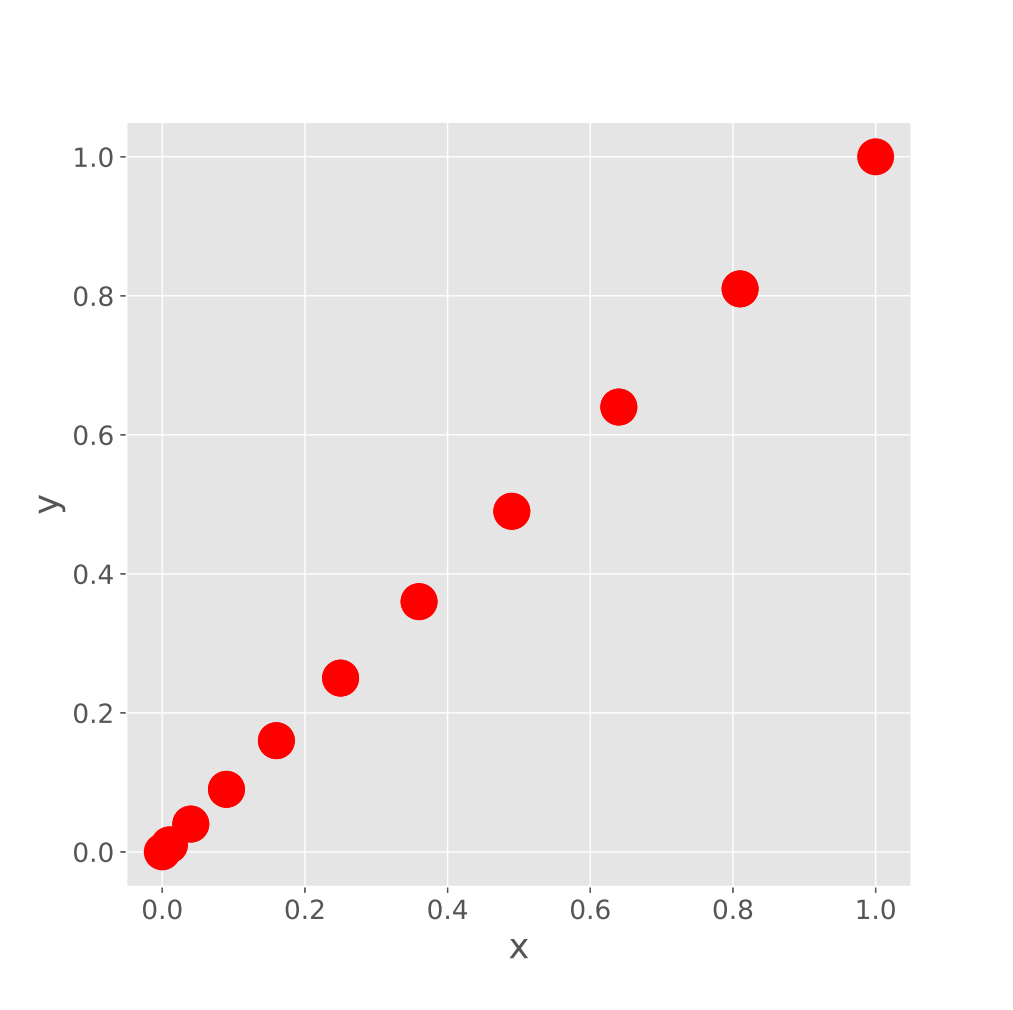
- A aplicação de uma função não-linear, se escolhida adequadamente, pode gerar novos atributos que possuem relação linear com a variável alvo.
- Funções comumentes utilizadas para essa tarefa:
  - **Sigmoid:** $\frac{1}{1+e^{-x}}$ cria uma variável em um tipo sinal, com valores entre 0 e 1.
  - **Tangente Hiperbólica:** $\tanh(x)$ idem ao anterior, mas variando entre -1 e 1.
  - **Logaritmo:** $\log(x)$ lineariza variáveis que seguem uma lei de potência.
  - **Rectified Linear Units:** $\max(0, x)$, elimina os valores negativos, relacionado com Redes Neurais.
  - **Softmax:** $\frac{e^{x_i}}{\sum_j e^{x_j}}$, similar a sigmoid, mas também faz com que a soma dos valores de x seja igual a 1.

Interação

- A interação polinomial gera interações de grau p entre as variáveis.
- Por exemplo, em um problema com 2 variáveis e $p = 2$ teríamos as variáveis $x_1 \cdot x_1, x_1 \cdot x_2, x_2 \cdot x_2$.
- Já para $p = 3$ teríamos $x_1^3, x_1^2 \cdot x_2, x_1 \cdot x_2^2, x_2^3$.

Interação

- Introduzindo a variável x_1^2 em nosso exemplo, temos:



Outros algoritmos

- k-Vizinhos Mais Próximos
  - Sua adaptação para regressão é trivial, basta fazermos as médias dos valores $\{y_i\}_{i=1}^{k}$ dos k vizinhos
  - Média aritmética ponderada?

Kernel Regression

- Levar em consideração todos os pontos, mas ponderando a distância de todos
- Kernel é a função de ponderação
  - Gaussiano é comum: $w(q; x, \sigma) = e^{\frac{-(x-q)^2}{\sigma^2}}$
- Vantagens: kernel permite aproximar funções mais complexas

Kernel Regression: Exemplo

- Considere a base de dados:

x	y
1	1
2	2
3	4
4	8
5	4
6	2
7	1

- Com kernel: $w(q;x) = \frac{1}{(x-q)^2}$
- Como ponto de teste/consulta considere $q = 3.5$.

Kernel Regression: Exemplo

x_i	1	2	3	4	5	6	7
y_i	1	2	4	8	4	2	1
w_i	4/25	4/9	4	4	4/9	4/25	4/49
$w_i \cdot y_1$	4/25	8/9	16	32	16/9	8/25	4/49